the intrinsic evidence of the patent specification shows different computer architectures as the source of the native and guest instruction sets. (*See also* Smotherman Decl. ¶ 9.)

There can be no dispute that the computer discussed in the '520 patent supports *two distinct* instruction sets, one that is natively implemented and one that is emulated (the guest instruction set). ('520 patent, 1:45-2:28; claims 1 and 9.) Under PSI's construction, the instruction set "of a ['520] computer" would have to include a combination of both instruction sets, which would render the emulator unnecessary and would therefore make no sense. As a result, IBM's construction that an instruction set is "the complete set of the operations of the instructions of a computer architecture together with the types of meanings that can be attributed to their operands" is the proper construction.

2. "semantic routine" (claims 1, 4, 9, 12)

The only dispute the parties have with this term is whether a "semantic routine" is a "*set or sequence* of native instructions" (IBM) or a "*sequence* of native instructions" (PSI).

IBM's CONSTRUCTION	PSI's CONSTRUCTION
A defined *set or sequence* of native instructions that, when executed, emulates an associated guest instruction.	A defined *sequence* of native instructions that, when executed, emulates an associated guest instruction.

PSI's construction is incorrect because the plain and ordinary meaning of the term "sequence" in PSI's construction suggests a specific order of steps, and such a suggestion would unnecessarily limit the meaning of the term "semantic routine" in the '520 patent.

IBM does not dispute that a semantic routine could be a sequence of native instructions, but it is not limited to such a sequence. PSI cites no evidence that limits the '520 semantic routine to a sequence of native instructions. In the '520 patent, "guest instructions 20 are each emulated by fetching and executing one or more semantic routines 19, which each contain two or more native instructions." ('520 patent, col. 4:44-46.) Stated in an equally non-limiting fashion, "[s]emantic routine (i.e., native) instructions that are within the standard instruction set of CPU 4 are processed by CPU . . .." ('520 patent, col. 11:4-6.) Furthermore, as set forth in U.S. Patent

No. 5,732,235, incorporated by reference in the '520 patent and entitled "Method and System For Minimizing The Number of Cycles Required To Execute Semantic Routines," "[t]o enable the processor to emulate the guest instructions, each guest instruction is first translated into a *corresponding set of host instructions*, called a semantic routine, that perform the function of guest instruction in the host processor." (Ex. 36, '235 patent, col. 2:49-53, emphasis added.) While the '235 patent also refers to the semantic routines as "the sequence of host instructions necessary to perform the function of each guest instructions," in the absence of contrary evidence, IBM should be afforded the broadest possible meaning of the term. In this case, a semantic routine can be either a set or a sequence of native instructions.

PSI's proposal that a routine requires a sequence is also contrary to the plain and ordinary meaning of the word "routine." The common definition for "routine" is a standard practice or regular course of procedure. (Ex. 37, *Webster's Third New International Dictionary,* p. 1981 (1993).) For example, if one's morning routine as to do "X, Y, and Z" before heading to work, it is no less of a routine if one does "Y, X, and Z" one day and "Z, X, and Y" the next. In computer science, a routine is "a generic term for any section of code that can be invoked (executed) within a program." (Ex. 38, *Microsoft Press Computer Dictionary*, 2nd ed., p. 344 (1994).) Likewise, the computer science definition of routine does not limit it to a sequence of code that is invoked. (*See* Smotherman Decl. ¶¶ 24-25.)

For these reasons, IBM's construction that a "semantic routine" is a "a defined set or sequence of native instructions that, when executed, emulates an associated guest instruction" is the proper construction.

3. "means, responsive to receipt of said guest memory access instruction for emulation, for translating said guest logical address into a guest real address and for thereafter translating said guest real address into a native physical address" (claim 9)

The parties agree this is a means-plus-function element governed by 35 U.S.C. § 112, ¶ 6.

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Function: Translating said guest logical address into a guest real address and for thereafter translating said guest real address into a native physical address. Structure: A data processing system (Figs. 1, 2, or 3) configured to translate the guest logical address into a guest real address and thereafter to translate the guest real address into a native physical address (Figs. 7 or 8; column and lines: 13:52-14:45; or 14:46-15:7; or 15:56-65), and equivalents thereof.	Function: Translating said guest logical address into a guest real address and for thereafter translating said guest real address into a native physical address. Structure: Figs. 7, 8, 9.

Structure. The structure required to perform the function of this element is adequately set forth in the figures and specification of the '520 patent. Initially, a data processing system depicted in Figures 1, 2, or 3 would carry-out this function. For instance, "Figure 1 depicts an illustrative embodiment of a data processing system with which the method and system of the present invention may advantageously be utilized." ('520 patent, col. 3:42-44.) Figures 2 and 3 similarly set forth adequate structure to one of ordinary skill in the art to illustrate a data processing system required to implement this function.

Structure for both "translating said guest logical address into a guest real address" and "thereafter translating said guest real address into a native physical address" components of this function are depicted in Figures 7 or 8, and in column and lines 13:52-14:45, or 14:46-15:7, or 15:56-65. Figure 7 depicts how each guest logical address is first translated into a guest real address (using the logical equivalent of the address translation scheme depicted in Figure 6), and is thereafter translated into a native physical address (using the logical equivalent depicted in Figure 5). ('520 patent, col. 13:35-51.) Figure 8 depicts a high level flowchart that illustrates how the process of guest address translation shown in Figure 7 is implemented. *Id.*, col. 13:52-55. PSI also cited Figure 9 for structure of this function, and IBM agrees it should be added as structure.

Other than Figures 7, 8, and 9, PSI fails to cite anything in the specification as structure for this function. However, the process of guest address translation begins in response to receipt of a guest linear address by the memory management unit. ('520 patent, col. 13:60-67.) Next, a storage area (DTLB) determines whether or not the translations have already occurred. *Id.*, col. 13:67-14:17. If the guest linear address has not been translated, an exception handler makes use of the logical equivalent of the address translation process of Figure 6 to generate a guest real address from the guest linear address. *Id.*, col. 14:18-41. Subsequently, the guest real address is translated into a native effective address by adding an offset value, which is then loaded in the storage area and translated into a native physical address. *Id.*, col. 14:41-15:7. Thus, the specification specifically "links or associates" the structure cited by IBM to this claimed function.

4. "means for executing a semantic routine that emulates said guest memory access instruction utilizing said native physical address" (claim 9)

The parties agree this is a means-plus-function element governed by 35 U.S.C. § 112, ¶ 6.

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Function: Executing a semantic routine that emulates said guest memory access instruction utilizing said native physical address. Structure: A data processing system (Figs. 1, 2, or 3) configured to execute a semantic routine that emulates the guest memory access instruction utilizing the native physical address (Fig. 8; column and lines: 11:4-15; 11:26-31; 13:60-14:25; 15:56-67), and equivalents thereof.	Function: Executing a semantic routine that emulates said guest memory access instruction utilizing said native physical address. Structure: Figs. 7, 8, 9.

Structure. The structure required to perform the function of this element is set forth in the figures and specification of the '520 patent. A data processing system depicted in Figures 1, 2, or 3 would implement this function. Figure 1 is an illustration of a data processing system in which the method and system of the invention is employed, in particular executing the semantic

routines that emulate the guest memory access instructions using native physical addresses. ('520 patent, col. 3:42-44.) Figures 2 and 3 similarly set forth adequate structure to one of ordinary skill in the art for a data processing system required to implement this function.

As explained in the specification, semantic routine (*i.e.*, native) instructions within the standard instruction set of CPU 4 are processed by CPU 4 as described above with reference to FIG 2. ('520 patent, col. 11:4-6.) Guest instructions are connected in a continuous stream by inserting a special native instruction into the end of each semantic routine. *Id.*, col. 11:9-14. Once the effective address of the semantic routine corresponding to the next guest instruction is computed, that semantic routine is fetched from memory for execution by the CPU. *Id.*, col. 11:26-31. When executed, the semantic routines emulate a guest memory access instruction using native physical addresses. *Id.*, col. 13:61-14:17. Thus, the specification specifically "links or associates" the structure cited by IBM to this claimed function.

PSI contends that the structure for the function "executing the semantic routines..." is set forth in Figures 7, 8, and 9. IBM agrees that Figures 7 and 8 depict adequate structure for this function. However, Figure 9 is a diagram of certain registers used in the address translation scheme illustrated in Figures 7 and 8. ('520 patent, col. 3:61-62.) This figure is structure for the "means for translating" function and not, as PSI contends, for the "executing the semantic" function.

B. The '261 "Emulation" Patent

The '261 patent discloses a method for emulating incompatible instructions on a processor ("the target processor") that would not otherwise be capable of executing those instructions. Target instructions within a target routine are preprocessed by a set of patching instructions to enable execution of the incompatible instructions on the target processor. The parties dispute the construction of the terms "processor" (discussed above), "patching instruction," "incompatible instruction," "target instruction," and "target routine."

1. "patching instruction" (claims 1, 2, 7, 8, and 10)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
An element of a target routine that is used to copy or modify some or all of a target instruction, to enable a guest instruction to be emulated.	An element of a target routine that is used to copy or modify some or all of a target instruction, to enable a guest instruction to be emulated in a dynamic manner each time a guest instruction is encountered.

The parties agree that a "patching instruction" is "an element of a target routine that is used to copy or modify some or all of a target instruction, to enable a guest instruction to be emulated." Nothing more is necessary to define a patching instruction.

PSI's construction unnecessarily adds that the *emulation* is done "in a dynamic manner each time a guest instruction is encountered." The additional language in PSI's proposed construction has nothing to do with the term "patching instruction," but instead relates to the general concept of instruction translation described in the beginning of the '261 patent. The full description of the limitation PSI improperly attempts to insert into the construction states: "By doing the instruction translation in a dynamic manner each time an incompatible instruction is encountered, it is unnecessary to save the translations for future use, saving the storage required to preserve these in target storage." ('261 patent, col. 3:67-4:4.) This passage refers to one embodiment for one type of dynamic emulation and not a patching instruction. If the inventors wanted or needed to limit the claims to a dynamic type of emulation, they could have specifically done so in the claims. They chose not to do so.

By appending this language to the phrase "to be emulated" in its proposed construction, PSI is using its construction for the term "patching instruction" to further limit the term "emulation," a previously agreed-upon claim term among the parties. The parties already agreed that "emulation" means:

> Using a data processing system to imitate another data processing system, so that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system.

(Ex. 35, Exhibit A of the Joint Statement, p. 4.) The parties' definition of "emulation," taken directly from the *IBM Dictionary of Computing*, does not reference dynamic emulation. By adding "in a dynamic manner each time a guest instruction is encountered," PSI is doing nothing more than trying to further limit the term "emulation." Once again, PSI is improperly attempting to add limiting language to support a noninfringement argument.

Accordingly, the construction for "patching instruction" should be properly construed as "an element of a target routine that is used to copy or modify some or all of a target instruction, to enable a guest instruction to be emulated." Furthermore, it is not necessary to define the manner of emulation within the construction of this term – the parties have already agreed on a construction for "emulation."

2. "incompatible instruction" (claims 1, 7, 8, and 10) and "target instruction" (claims 1, 2, and 7-12)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Incompatible Instruction - A language construct that specifies an operation and identifies its operands, if any, and pertains to the architecture being emulated, which is different than the architecture on which the emulator runs. Target Instruction - A language construct that specifies an operation and identifies its operands, if any, and pertains to the architecture on which the emulator runs.	Incompatible Instruction - A string of digits that specifies an operation and identifies its operands, if any, and would be able to be directly executed by a processor of the emulated data processing system. Target Instruction - A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed by the processor to which it is directed.

The dispute regarding these terms relates to the parties' disagreement regarding the meaning of "instruction," discussed above in Section V. PSI's proposed constructions, which both require that the instruction be "directly executed by a processor," add limitations not supported by the intrinsic or extrinsic evidence.

IBM's construction for these terms can be broken down into two parts – the "incompatible/target" part (grammatically second in the constructions) and the "instruction" part. The parties have already agreed on constructions for incompatible and target:

incompatible - pertaining to the architecture that is being emulated.

target - pertaining to the architecture one which the emulator runs.

(Ex. 35, Exhibit A of the Joint Statement, p. 4.) IBM incorporated the above agreed-upon definitions into its constructions for "incompatible instruction" and "target instruction." Rather than use these agreed-upon terms, PSI seeks to add unsupported limitations in its constructions by referencing a processor that would "directly" execute the instruction.

PSI's proposed construction for "incompatible instruction" would require the instruction to "be able to be directly executed by a processor of the emulated data processing system." Likewise, PSI's proposed construction for "target instruction" would require the instruction to "be directly executed by the processor to which it is directed." PSI is adding the language relating to direct execution with no support whatsoever from the intrinsic evidence. Instead, it is relying on a definition for "machine instruction" from the *IBM Dictionary of Computing*, which, as discussed in Section V, is different from the term "instruction." Nothing in the intrinsic evidence references a machine instruction in particular. PSI's attempt to limit the constructions in this manner is contrary to the law and should not be accepted.

3. "target routine" (claims 1, 2, 7-9, 11, and 15)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
The set or sequence of target instructions that enable an incompatible instruction to be run on target computer system.	A defined sequence of target instructions performing a function similar to, but not necessarily identically to, a corresponding incompatible instruction.

The parties have the same dispute for this term as they do for the term "semantic routine" in the '520 patent, which is, whether a "routine" is a "*set or sequence* of target instructions" (IBM) or a "defined *sequence* of target instructions" (PSI). *See Section* VII A, *supra*. In addition, the parties dispute the remainder of the construction insofar as PSI's construction is redundant and offers no additional meaning beyond that provided by the claim language.

Routine - Set or Sequence

Instead of re-stating IBM's full position regarding why a "routine" should be construed as either a set or sequence of target instructions, IBM directs the Court to Section VII A of this brief regarding this term for the '520 patent. It is worth noting, however, that nowhere in the claim language or specification of the '261 patent is the term "target routine" limited to a "defined sequence of target instructions" as proposed by PSI. In fact, the specification states, "each incompatible instruction instance is translated to a *set* of equivalent target instructions in a corresponding target translation routine for execution in the target processor. ('261 patent, col. 9:27-30, emphasis added.) By limiting its construction for "target routine" in the '261 patent to just a sequence, as PSI does for its construction of "semantic routine" in the '520 patent, PSI is improperly limiting the scope of the claims in the '261 patent.

To Enable an Incompatible Instruction to be Run on Target Computer System

IBM's construction of a target routine gives the appropriate description for the meaning of the term. The language PSI proposes after "a defined sequence of target instructions" is redundant given the entirety of the claim language. For example, PSI's construction in the context of claim 1 would read (PSI construction shown in []):

> and each [defined sequence of target instructions *performing a function similar to, but not necessarily identically to, a corresponding incompatible instruction*] performing a function similar to, but not necessarily identically to, a corresponding incompatible instruction.

Contrast this with IBM's construction (shown in []), and it is clear IBM's construction is proper:

> and each [set or sequence of target instructions that enable an incompatible instruction to be run on target computer system] performing a function similar to, but not necessarily identically to, a corresponding incompatible instruction.

Consequently, IBM's construction that a "target routine" is "the set or sequence of target instructions that enable an incompatible instruction to be run on target computer system" is proper given the plain and ordinary meaning of the terms, claim language, and specification.

VIII. THE FLOATING POINT PATENTS

IBM mainframes perform various operations, including "floating point" calculations, in specific ways. In order for PSI's system to accept the same data, execute the same programs, and achieve the same results as the IBM systems it attempts to imitate, PSI must mimic the floating point operations of IBM's systems. Several of the asserted patents relate to the manner in which IBM mainframes perform those calculations.

There are limitations on the number of digits that computers can use to represent data. In a system that is limited to eight digits, 99999999 is the largest available value and 00000001 is the smallest. In addition, a decimal point can be added to the representations of numbers, which would then make 9999999.9 the largest number. This type of notation is known as "fixed point notation." In order to manipulate a larger number using the same eight digits, computers also use what is known as "floating point notation." With floating point notation, the number is not stored as a string of digits, but rather has three components: (1) a fraction multiplied by the appropriate base (*i.e.*, base 10 for decimal notation); (2) that is raised to a given power; and (3) the sign of the number as positive or negative. Decimal examples include $+6.02 \times 10^{23}$ or -9.109×10^{-31}.

Most computer systems, including PSI's infringing system, use floating point numbers with a base other than "10." Floating point numbers with a base of "2" (binary) or "16" (hexadecimal) are most common.

In this case, IBM is asserting three patents that deal with floating point operations.

A. The '709 "Rounding Mode" Patent

The parties dispute two terms in the asserted claims of the '709 patent, "processor" and "instruction." Both of these terms are addressed in Section V above. Thus, no disputed terms are discussed here. IBM will nevertheless provide a brief description of this patent.

During mathematical computation by a computer system, it is often desirable to round a floating point number. Depending on the application and the context, different rounding techniques, known as "rounding modes," can be used. For example, some applications might

round to the nearest integer, so that 11.2 and 10.8 would both round to 11. Other applications might always round upward, so 11.2 would round to 12. The rounding options expand further when negative numbers are considered.

Thus, it becomes necessary to be able to set which "rounding mode" should be used by the computer system for particular operations. Some systems allow programmers to specify a rounding mode for all operations until a change is indicated, while others require programmers to specify a rounding mode for each individual operation. The '709 patent discloses a system that allows programmers to go either route, providing a configurable "default rounding mode" that can be used from operation to operation as well as giving the programmer the option to indicate a different mode for a particular operation without changing the overall default mode.

B. The '678 "Data Class" Patent

Floating point numbers can be classified into one or more floating point data classes based on the sign, fraction, and exponent of the number (*e.g.*, the positive number class and the negative number class). Programs are often designed to treat floating point numbers differently based on the class, rather than the precise value, of a number. For example, financial applications often display negative numbers in red and positive numbers in black. More complex programs may have more complex requirements. Positive and negative results might be handled one way, while zero or an infinitely large result might be handled another way.

The '678 patent teaches a Test Data Class Instruction for a computer system to consider multiple discrete classes essentially as one combined class, so that a programmer can simply determine whether a number is "positive or negative" or "zero or infinite" rather than having to separately check each case.

For the '678 patent, the parties dispute the construction of four claim terms. Two of the terms, "processor" and "program status word" are discussed above in Section V. Two other terms in dispute are "floating point processor" and "a machine instruction." The parties also dispute the corresponding structure recited in the '678 specification for the means-plus-function elements of claim 1.

1. "a floating point processor" (claim 1)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
A portion of a computer system that interprets and executes floating point instructions.	A processor that contains a floating point unit. A floating point unit is that portion of a processor's circuitry that executes floating point instructions by performing operations on floating point numbers.

The computer systems at issue in this case have separate processors that deal with floating point operations. IBM's construction of "floating point processor" is a simple modification of its construction for "processor" ("a portion of a computer system that interprets and executes instructions") to reflect that the floating point processor works with floating point instructions. IBM's proposed construction of "processor" is discussed in Section V above.

IBM's construction of "processor" is based on IBM's Dictionary of Computing, which defines a processor as follows: "[i]n a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit." (Ex. 26, *IBM Dictionary of Computing*, 10th Ed., p. 533 (1994).) Similarly, the plain and ordinary meaning of "*floating point* processor" is "a portion of a computer system that interprets and executes *floating point* instructions." This is supported by the '678 claims and the '678 specification. For example, the '678 specification describes components of the CPU (processor) according to the invention as being capable of "appropriately effect[ing] execution of these instructions" ('678 patent, col. 2:43-44) and being able to "execute the floating point instruction set." *Id.*, col. 2:48-49. More importantly, claim 1 itself recites "a floating point processor for interpreting a machine instruction." *Id.*, col. 4:63-64.

PSI's construction attempts to insert a definition of "floating point unit" and its circuitry into the construction of "floating point processor." In doing so, PSI improperly inserts limiting language from the '678 specification into the claims. *See Phillips*, 415 F.3d at 1323 ("Although the specification often describes very specific embodiments of the invention, we have repeatedly warned against confining the claims to those elements.").

A floating point processor can include a floating point unit, but the term "floating point unit" never appears in the claims and is not a term that needs construction. The '678 specification and prosecution history do not state that a floating point processor *must* include a floating point unit, thus leaving open the possibility that a floating point processor can interpret and execute floating point instructions in other embodiments that do not include a floating point unit.

Further, PSI's definition of floating point unit, and in particular the limiting language that it must be a "portion of the processor's circuitry," is not required by the '678 intrinsic evidence. While the '678 specification states that "[f]loating point unit 204 includes all necessary hardware to execute the floating point instruction set," ('678 patent. col. 2:47-48) this applies to the preferred embodiment described by the specification, and the '678 claims, specification, and prosecution history do not limit the floating point unit (or, for that matter, the floating point processor) to hardware only.

Because the intrinsic and extrinsic evidence do not limit "floating point processor" to "floating point unit" or circuitry, PSI's construction should be rejected.

2. "a machine instruction" (claims 1, 3, 7, 9)

IBM's construction of the term "machine instruction" is similar but not identical to its construction of the term "instruction," discussed above in Section V. The only difference is that IBM's construction adds that a machine instruction "can be directly executed by a processor of a computer." PSI, on the other hand, provides identical constructions for the terms. PSI's approach is contrary to the plain and ordinary meaning of these terms.

IBM's CONSTRUCTION	PSI's CONSTRUCTION
A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed *by a processor of a computer.*	A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed *by the processor to which it is directed.*

There is no specific language in the patent that spells out the plain and ordinary meaning for machine instruction, other than the claim language itself. The plain and ordinary meaning is explained in the Declaration of IBM's expert, Dr. Smotherman. (*See* Smotherman Decl. ¶¶ 9-12.) In addition, IBM relies on the definitions of "instruction" and "machine instruction" in the *IBM Dictionary of Computing*. It defines an "instruction" as "[a] language construct that specifies an operation and identifies its operands, if any," and a "machine instruction" as "a[n] instruction that can be directly executed by a processor of a computer." (Exs. 28, 29.)

The parties' proposed constructions are similar, with the exception of the one difference shown in italics in the chart above (IBM is willing to agree to PSI's introductory phrase "a string of digits" that it previously disputed). IBM's proposed construction includes that the string of digits "can be directly executed by a processor of a computer," which as discussed above is based on the dictionary definition of "machine instruction" in the *IBM Dictionary of Computing*.

PSI's proposed construction, on the other hand, states that the "string of digits" "can be directly executed by *the processor to which it is directed*." As discussed above in Section V, PSI changes the language from the *IBM Dictionary of Computing*, presumably in an attempt to avoid infringement. But there is nothing in the intrinsic or extrinsic evidence that supports such a construction.

3. "means for retrieving the floating point number from memory" (claim 1)

The parties agree this is a means-plus-function element governed by 35 U.S.C. § 112, ¶ 6.

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Function: Retrieving the floating point number from memory. Structure: A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for retrieving the floating point number from memory (*see, e.g.*, Fig. 8, 3:63-66), and equivalents thereof.	Function: Retrieving the floating point number from memory. Structure: No corresponding structure.

Structure. The '678 specification discloses the corresponding structure in the form of a shared memory computer system. (*See* '678 patent, Fig. 1, col. 2:35-38.) The '678 specification also discloses the execution of instructions by the shared memory computer system:

> Fig. 2 illustrates functional components included in a CPU from Fig. 1. Instruction unit 200 fetches instructions from common main storage ~~110~~ according to an instruction address located in the program status word (PSW) register 202, and appropriately effects execution of these instructions. Instruction unit 200 appropriately hands off retrieved floating point instructions to floating point unit 204, along with some of the operands that may be required by the floating point unit to execute the instruction...FP unit 204 is coupled to floating point (FP) registers 206, which contain floating point operands and results associated with FP unit 204 processing, and is also coupled to general registers 208.

('678 patent, col. 2:39-54; *see also* Fig. 2.)

The '678 specification next discloses (and illustrates in Fig. 5) a "typical instruction format," and that the field "R1 designates the floating point register to be tested." ('678 patent, col. 3:27-28.)

The '678 specification later discloses (and illustrates in Fig. 8) the remaining algorithm for retrieving the floating point number from memory:

> The floating point number in the floating point register that is pointed to by R1 is loaded into the convert to type number logic and a determination is made as to the data class of the number.

('678 patent, col. 3:63-66.) Thus, the '678 specification discloses a shared memory computer system and the algorithm for retrieving a floating point number from memory.

PSI again erroneously asserts that there is no corresponding structure for this element. As set forth above, this is incorrect, as there is ample disclosure in the specification linking the structure cited by IBM to this function.

Accordingly, IBM's proposed structure is consistent with the intrinsic evidence, corresponds to the recited function, and should be adopted.

4. "means for determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number" (claim 1)

The parties agree this is a means-plus-function element governed by 35 U.S.C. § 112, ¶ 6.

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Function: Determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number. Structure: A shared memory computer system (*see*, *e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number (*see*, *e.g.*, Fig. 5, 3:34-45; Fig. 7, 3:49-50; Fig. 8, 3:66-4:5; Fig. 9, 4:9-32), and equivalents thereof.	Function: Determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number. Structure: Fig. 9, except for the circuitry that connects AND gates to the box marked CC in Fig. 9 (known in the art as a "wired OR gate").

Structure. The '678 specification also discloses the structure to perform this computer-implemented function. In addition to the shared memory computer system described for the previous element, the '678 specification discloses an algorithm for determining whether the data class of the floating point number is an identified data class.

The '678 specification primarily discloses this algorithm with Figs. 8 and 9. Fig. 8 illustrates the floating point number in the "convert to type number logic" based on retrieving the floating point number from memory. (*See* '678 patent, col. 6:63-66.)

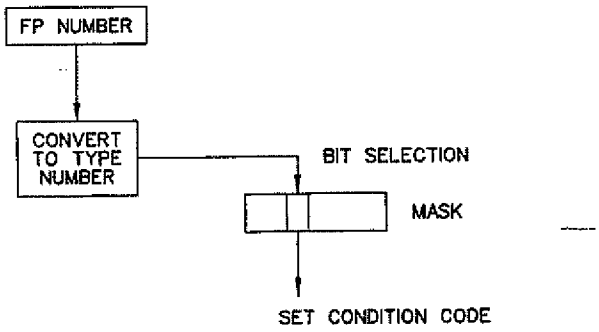

FIG.8

The '678 specification then states that "[i]n the convert to type number logic, these floating point numbers are classified according to the sign, the fraction part, and the exponent part of the floating point number. That is, the floating point format has sign, exponent, and fraction bit. Based on these three bits, the floating point number may be categorized [into a particular data class]." ('678 patent, col. 3:66-4:5.) By comparing this categorized information to the floating point data classes (*see id.*, Fig. 6), one can determine whether the data class is an identified data class.

Fig. 9 completes the algorithm by showing how each portion of the floating point number (sign, fraction part, and exponent part) is categorized. ('678 patent, col. 4:9-32.)

Thus, the '678 specification discloses the shared memory computer system and the algorithm for determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number.

PSI asserts that the corresponding structure is limited to a part of Fig. 9. As set forth above, PSI's proposed structure is incomplete, because other parts of the '678 specification (*e.g.*, '678 patent, Figs. 1, 2, and 8; col. 2:35-59, 3:66-4:32) describe the shared memory computer system and the corresponding algorithm for performing the recited function.

Accordingly, IBM's proposed structure is consistent with the intrinsic evidence, corresponds to the recited function, and should be adopted.

5. "means for setting a condition code in a program status word based upon the determination of whether the data class is the identified data class" (claim 1)

The parties agree this is a means-plus-function element governed by 35 U.S.C. § 112, ¶ 6.

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Function: Setting a condition code in a program status word based upon the determination of whether the data class is the identified data class. Structure: A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for setting a condition code in a program status word based upon whether the data class is the identified data class (*see, e.g.*, Fig. 8, 4:5-8; Fig. 9, 4:9-32), and equivalents thereof.	Function: Setting a condition code in a program status word based upon the determination of whether the data class is the identified data class. Structure: The circuitry that connects the AND gates to the box marked CC in Fig. 9 (known in the art as a "wired OR gate").

Structure. The '678 specification discloses the structure to perform this computer-implemented function. In addition to the shared memory computer system described above, the '678 specification discloses an algorithm for setting a condition code in a program status word based upon the determination of whether the data class is the identified data class. For example, Fig. 8 (shown above) illustrates the determination of the floating point data class and the "BIT SELECTION" for the bit mask. As the specification explains:

> Based on [the sign, exponent, and fraction bit], the floating point number may be categorized. Based upon the particular data class that it falls into, a signal is generated that will indicate one of the bits in the mask. That bit is then loaded into the condition code. That is, the condition code is set.

('678 patent, col. 4:4-8.) Fig. 9 then illustrates one way to set the condition code. The '678 specification describes Fig. 9 and explains in detail how the bit mask is coded and the condition code is set. *Id.*, col. 4:9-32.

Thus, the '678 specification discloses the shared memory computer system and the algorithm for setting a condition code in a program status word based upon the determination of whether the data class is the identified data class.

PSI again asserts that the corresponding structure is limited to another part of Fig. 9. As set forth above, PSI's proposed structure is incomplete, because other parts of the '678 specification (*e.g.*, '678 patent, Figs. 1, 2, and 8; col. 2:35-59, 3:66-4:32) describe the shared memory computer system and the corresponding algorithm for performing the recited function.

Accordingly, IBM's proposed structure is consistent with the intrinsic evidence, corresponds to the recited function, and should be adopted.

C. The '106 "Floating Point Conversion" Patent

Different computer systems have implemented floating point arithmetic in different ways, such as by using different bases (base 2, base 10, or base 16), and different ways of storing the numbers in memory. IBM's '106 patent describes a computer system that is compatible with multiple "floating point architectures." The patent teaches the conversion of floating point numbers from different architectures to a single internal format. The part of the computer system that performs the floating point calculations, known as the "floating point unit," uses this internal format for floating point calculations. After the floating point calculations are completed in the internal format, the results are converted back from the internal format to the original notation. This allows a computer system to be compatible with multiple floating point architectures without requiring a separate floating point unit for each supported architecture.

The parties dispute the construction of the claims terms "floating point unit," "a floating point unit having an internal dataflow," "(a floating point unit that) supports both said first floating point architecture and said second floating point architecture," and "converter."

1. "floating point unit" (claims 11 and 12)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Part of a computer system that performs floating point operations.	That portion of a processor's circuitry that performs floating point calculations.

The parties' primary dispute is whether the floating point unit is "part of a computer system" (IBM's construction) or a "portion of the processor's circuitry" (PSI's construction).

IBM's construction is based on the claims and specification of the '106 patent, and particularly claim 11, which recites that the floating point unit is part of a computer system ("[i]n a computer system, a floating point unit that"). ('106 patent, col. 22:7.) Contrary to PSI's construction, the '106 claims do not limit the floating point unit to a portion of the processor's circuitry.

In addition, the '106 specification illustrates a floating point unit in Fig. 1, and states that "Fig. 1 provides an overview of the dataflow of the Floating Point Unit (FPU) which supports both S/390 and IEEE 754 architecture." ('106 patent, col. 3:53-55.) PSI will likely cite to the subtitle in column 3: "Overview of Hardware Support for Multiple Architectures" as a limiting disclosure. But even that heading does not limit the disclosed structure to hardware – a person of ordinary skill in the art would understand that the components described in this section can be implemented in hardware, software, or a combination of both. (*See* Smotherman Decl. ¶¶ 16-18.) In fact, the '106 inventors recognized this when they included the following language at the end of the '106 specification:

> Although the above description provides many specificities, these enabling details should not be construed as limiting the scope of the invention, and it will be readily understood by those persons skilled in the art that the present invention is susceptible to many modifications, adaptations, and equivalent implementations without departing from this scope and without diminishing its attendant advantages. It is therefore intended that the present invention is not limited to the disclosed embodiments but should be defined in accordance with the claims which follow.

('106 patent, col. 20:47-56.)

With respect to the parties' use of the competing phrases "portion of a processors circuitry" (PSI) versus "part of a computer system" (IBM), these phrases differ in two meaningful respects. First, PSI's language limits the location of the floating point unit to being part of a "processor," as opposed to being part of a "computer system." As discussed above the '106 claims plainly indicate that the floating point unit is part of a computer system, and the term "processor" that PSI is now attempting to read into the claim does not appear in the specification. Moreover, the specification reveals no reason why the floating point unit must be part of a "processor" as opposed to some other part of the computer system.

Second, PSI's language unnecessarily limits the claimed floating point unit to being "circuitry." Again, "circuitry" never appears in the specification, and the specification provides no reason why the floating point unit must be implemented in circuitry, as opposed to some other implementation such as a partial software implementation.

With respect to extrinsic evidence, PSI selectively cites dictionary definitions that cannot trump the plain language of the '106 claims and '106 specification.

The parties also disagree about whether the floating point unit performs floating point "operations" (IBM's construction) or floating point "calculations" (PSI's construction). Claims 1 and 11 of the '106 patent provide guidance, stating that "said floating point unit performing floating point *operations*," ('106 patent, col. 20:65-66, claim 1, emphasis added) and that the "floating point unit. . . performs floating point *operations* on the data formatted in said internal floating point format." *Id.*, col. 22:14-16, claim 11, emphasis added. Similarly, the '106 specification repeatedly refers to "operations" and "floating point operations." *See, e.g., id.*, col. 1:58-63; 3:20-24; and 4:60-62.

2. "a floating point unit having an internal dataflow" (claim 11)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
A floating point unit which includes the functions of moving data through the unit and executing floating point operations.	A floating point unit that uses an internal floating point format to perform floating point calculations.

Because "floating point unit" was discussed above, the remaining part of this phrase that requires a construction is "internal dataflow."

Based on its plain and ordinary meaning, the internal dataflow of the floating point unit is the internal flow of data through the unit. The definition is confirmed by definition of "data flow" in the *Microsoft Press Computer Dictionary*: "the movement of data through a system from its entry to its destination." (Ex 39, *Microsoft Press Computer Dictionary*, 2nd Ed., p. 107 (1994).) The first part of IBM's proposed construction ("moving data through the unit") captures this notion.

According to the '106 specification, one embodiment of the invention includes "an internal dataflow which *executes all floating point operations* using an expanded format that accommodates both floating point formats." ('106 patent, col. 3:22-24, emphasis added). The second part of IBM's construction ("executing floating point operations") captures this aspect of the internal dataflow.

PSI's proposed construction for the disputed phrase is "a floating point unit that uses an internal floating point format to perform floating point calculations." But the disputed phrase is immediately followed in claim 11 by the language "having an internal floating point format (*see* '106 patent, col. 20:61-62), and PSI's proposed construction would add redundant language – in the context of claim 11 PSI's construction would read "a floating point unit that uses an internal floating point format to perform floating point calculations having an internal floating point format." IBM's proposed construction avoids this unnecessary redundancy.

3. "(a floating point unit that) supports both said first floating point architecture and said second floating point architecture" (claim 11)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Provides the necessary resources for the correct operation of both the first and second floating point architectures.	A floating point unit "supports" a floating point architecture when it can, on its own, perform calculations on floating point numbers of that architecture.

IBM's construction of this phrase is consistent with its plain and ordinary meaning and with the intrinsic evidence. The construction turns on the word "supports." The '106 patent describes a computer system that supports two different floating point architectures by converting those architectures to a common internal format, performing the necessary floating point operations in the internal format, and converting back to the original architectures. But the '106 claims, specification, and prosecution history need not provide a definition for "supports," because the plain and ordinary meaning controls. There is certainly no support in the '106 intrinsic evidence for PSI's proposed construction, and in particular that a floating point unit only "supports" a floating point architecture when it "on its own" performs calculations on floating

point numbers in that architecture. The '106 specification describes a particular implementation of a floating point unit that supports multiple floating point architectures by using the conversion method described above, and not by addressing a single architecture "on its own."

IBM's proposed construction is supported by the *IBM Dictionary of Computing's* definition of "support":

> In system development, to provide the necessary resources for the correct operation of a functional unit.

(Ex. 40, *IBM Dictionary of Computing*, 10th Ed., p. 663 (1994).) This construction takes into account the conversions and internal format of the floating point unit. Accordingly, the Court should adopt IBM's proposed construction.

4. "converter" (claim 11)

IBM's CONSTRUCTION	PSI's CONSTRUCTION
Hardware and/or software that changes data from one format or architecture type to another format or architecture type.	Circuitry within the floating point unit that changes the format or architecture type of a floating point number.

The two primary disputes between the parties are: (1) whether a "converter" must be limited to "circuitry" and (2) whether the converter must be located "within the floating point unit."

The '106 intrinsic evidence does not limit the term "converter" to "circuitry," and clearly allows for part of the converter to be implemented in software. The '106 claims, which are accorded the most weight, do not limit the converter to hardware or software and are broad enough to cover both. (*See, e.g.*, '106 patent, claim 11, col. 22:6-34.) The '106 specification also does not limit the term "converter" to hardware (or software), and in fact provides examples of both. For instance, the '106 specification describes one embodiment with "4 types of format conversion hardware." *Id.*, col. 3:50. The '106 specification also describes conversions "performed by a *software routine*" for extended operands. *Id.*, col. 7:57-59, emphasis added. The '106 inventors incorporated flexibility into the specification by not limiting the claims to the

disclosed embodiments and expressly stating that a person of ordinary skill in the art would understand equivalent implementations, such as using hardware, software, or a combination of the two. *See Id.*, col. 20:47-56.

The extrinsic evidence also supports IBM's proposed construction. Dictionary definitions of "data converter" and "converter" from the *IBM Dictionary of Computing* do not limit a converter to circuitry. For example, the IBM definition of data converter is "[a] functional unit that transforms data from one representation to an equivalent representation," (Ex. 41, *IBM Dictionary of Computing*, 10th Ed., p. 168 (1994)), while the IBM definition of converter is "[a] device that can convert impulses from one form to another, such as analog to digital, parallel to serial, one code to another, or one protocol to another." (Ex. 42, *IBM Dictionary of Computing*, 10th Ed., p. 149 (1994).) Thus, the term "converter" as used in the '106 patent is not limited to "circuitry" as PSI asserts, but rather can be a hardware and/or software implementation.

Similarly, PSI's attempt to limit a converter to "circuitry *within the floating point unit*" is completely contrary to the '106 intrinsic evidence, and should be rejected by the Court. Independent claim 11 of the '106 patent recites the "floating point unit" and each of the four "converters" as separate elements, with no language limiting the location of any of the converters within the floating point unit. Because the "converters" are recited as separate claim elements from the "floating point unit," the "converter" cannot be "within the floating point unit" as PSI proposes. In addition, although one embodiment of the '106 specification includes a set of converters within the floating point unit (*see* '106 patent, Fig. 1), the '106 specification does not expressly state (or even imply) that those converters must be located there.

Unlike PSI's proposal, IBM's proposed construction of "converter" is consistent with the '106 intrinsic and extrinsic evidence, and should be adopted.

IX. THE PARTITIONING PATENTS

A computer may be "partitioned," such that each "partition" acts as a separate computer, independently running its own operating system separate and apart from the other partitions. Further, a partitioned portion may itself be partitioned as well.